Combining policies: the best of human expertise and neurocontrol

Vincent Berthier, Adrien Couëtoux, Olivier Teytaud

Combining policies: the best of human expertise and neurocontrol

Vincent Berthier, Adrien Couëtoux, and Olivier Teytaud

TAO, Inria, Univ. Paris-Sud, UMR CNRS 8623
Bat 660 Claude Shannon Univ. Paris-Sud, 91190 Gif-sur-Yvette, France
firstname.lastname@inria.fr

Abstract. We consider sequential decision making in the case where a generative model and a parametric policy are available. Such a framework is naturally tackled with Direct Policy Search, i.e. parametric optimisation over simulations. We propose a simple method that combines this parametric policy with a more generic neural network, where all parameters are trained simultaneously. As such, our approach doesn't require any computational overhead. We show that the resulting policy significantly outperforms both the domain specific policies and the neural network on a unit commitment test problem.

1 Introduction

In this paper, we study planning under uncertainty, where only a generative model of the domain is available. We do not make any assumption on the inner dynamics of the problem.Instead, we assume that we have some prior knowledge, in the form of handcrafted parametric policies. These policies represent the existing methods to solve a problem. They can be optimal solutions of a simplified version of the problem, or simply human experience. The constants in those parametric policies are replaced by parameters optimised on simulations. This is Direct Policy Search, also known as Simulation-Based optimisation. More precisely, this is Direct Policy Search on top of expert policies; of course, Direct Policy Search can also be applied on top of generic policies such as neural networks or fuzzy rules. As Direct Policy Search rarely provides a gradient and needs a lot of robustness, it is usually optimized by evolutionary algorithms.

This approach is stable and efficient. It is particularly convenient when an expert policy is available [5]. However, in that case, it is limited by the structure of the policy. To combine and exploit existing solvers, portfolios are now a widely established principle. They are used in combinatorial optimisation [17, 11] and noisy optimisation [3], including applications to control [10]. In this work, we propose a simple method for combining parametric policies in a direct policy search framework. In contrast to portfolios as in [10], our solution not only selects the best of several policies but also in some cases vastly outperforms each of them, without computational overhead. We perform experiments on a unit commitment problem, a kind of power system management problem where the goal is to optimise the cost of energy production.

The following sections briefly review discrete time controls methodologies, surveys methods aimed at combining policies, and presents the concept of orthogonality in portfolios, which will be central in our work.

2 Background and notations

With states noted $x \in \mathcal{X}$ and actions $u \in \mathcal{U}$, we assume a generative model is available, *ie.* given (x, u), we can sample a resulting state $x' = f(x, u)$ and reward $r = \rho(x, u, x')$. f is the transition function and follows an unknown random distribution (e.g. $x' = f(x, u)$ depends on some random ω through $x' = f(x, y, \omega)$).

A policy π is an object that given a state x, returns an action u. It can be deterministic or stochastic, a parametric function or a qualitative heuristic. Note that if the problem is non-Markovian, optimal policies might require to include the entire history of observation in the state variable x, making methods sensitive to the size of the state space highly impractical [2].

The objective is to find a policy that maximizes the expected reward over a finite horizon T. Formally, given an initial state x_0, we try to find the solution π^* to

$$\underset{\pi}{\operatorname{argmax}}\, \mathbb{E}\left[\sum_t \rho(x_t, \pi(x_t), f(x_t, u_t))\right] \tag{1}$$

with $x_{t+1} = f(x_t, \pi(x_t))$ for $0 \leq t < T$.

2.1 Methodologies based on value functions

To find the optimal policy, the favourite methods in power systems applications (*eg.* the management of long term hydroelectricity storage en production), come from Dynamic programming [4] (DP) which is at the origin of a wide family of discrete-time control algorithms such as Stochastic Dual Dynamic Programming [18], Approximate Dynamic Programming [19], value iteration and a wide family of reinforcement learning algorithms. Despite their solid theoretical basis, they are computationally expensive, they cannot directly handle large scale non-Markovian random processes, and they are usually not anytime algorithms (*ie.* they return an incomplete answer if interrupted before termination). Because of this, they are often less efficient than simpler deterministic approaches [25, 7].

2.2 Direct Policy Search

Another trend in control is Direct Policy Search (DPS), which consists in searching in the policy space directly, without any proxy. This is often done by defining a set of parametric policies that depend on some parameter vector θ. One needs to find an optimal θ^*, so that π_{θ^*} is a solution to Eq. 1. The search for a good parameter θ can be done in a noisy optimisation framework, by relying on direct simulations of candidate policies π_θ on the test problem.

Various algorithms have been proposed, including evolutionary algorithms with resampling numbers chosen by Bernstein races [13] or by simple resampling rules [1]. They are often improved by the use of common random numbers [22, 23, 14].

The performance of parametric DPS heavily relies on the choice of the policy search space, i.e. the chosen class of policies that can be considered as candidates. Examples include neural networks [5] and fuzzy systems [24], usually optimised by evolutionary algorithms [21].

We use in this paper a self-adaptive evolution strategy, with anisotropic step-size [6]. The population size is set to $\lambda = 4N + 4$ where N is the number of parameters, and $\mu = \lambda/4$. The mutation rate is $\tau = 1/\sqrt{2N}$. Initial parameters are randomly drawn with a Gaussian distribution with step size 1 and step-sizes are independently randomly drawn as the exponential of standard Gaussian distributions.

3 Meta-policy search

To find an optimum solution, it is of course possible to try each of the policies, and select the best one. This however implies to run the optimisation process multiple times. Here, we propose a scheme to combine multiple policies: one is problem specific under the form of simple heuristics designed using prior knowledge on the domain, and the other one is a generic parametric policy (*eg.* Neural Network, Fuzzy rules).

3.1 Combining policies

Combining several policies has been done before, in different ways. A part of the literature combines policies in the sense that each policy, equipped with state prediction, handles a part of the state space [9]. Some approach combine policies based on their Q-functions [16] or by combining the policies themselves [12]. Another method is to distribute the computational power over a family of algorithms (similarly to how multi-armed bandits distribute arm pulls) by combining DPS algorithms [10].

As in [10], we consider a DPS-based approach. More precisely, we consider several parametric policies, to be optimised by DPS. However, instead of optimising each family separately, and then combining them, we consider a parametric combination $\alpha C_1 + (1 - \alpha)C_2$ where, C_1 and C_2 are two policies. We then optimise the joint policy. With this, the decision resulting from the joint policy is the combination of each policy's output.

More formally, given a current state s, we select the decision:

$$C_{combination}(s)\alpha C_1(s) + (1 - \alpha)C_2(s). \tag{2}$$

This makes sense in the case of continuous actions. The number of parameters to optimise is $N_1 + N_2 + 1$, where N_1 and N_2 are the number of parameters of

C_1 and C_2 respectively. We actually write α as a parametric function ranging from 0 to 1, with $\alpha = \frac{1}{2}(1+\beta/\sqrt{\beta^2+1})$; the parameter β is optimised in $\mathbb{R}$ and initialized at 0.

Our method has the following advantages:

- there is no computational overhead, as all the parameters of the combined policy are trained at once, without specific training of each independent policy. Most of the computation time is spent in the simulations, not in the policies themselves. Therefore the computational overhead for a given number of iterations, compared to each of the families separately, is negligible.
- we can outperform all the individual policies, as the global family of functions contains weighted averages of the original policies and not only the union of both families of functions.

3.2 Orthogonal policies

[20] pointed out the importance of using "orthogonal" algorithms in a portfolio. A portfolio containing too many optimisers tends to be unstable. It is then necessary to choose as few optimisers as possible, while covering as best as possible the set of all possible solvers. In order to increase the chances of finding a good solution, what matters is not (only) the number of optimisers in the portfolio, but how many orthogonal these optimisers are. Optimizers are said to be "orthogonal" if they are "very" different one to each other.

In the same way, combining many policies, or two policies of the same type (*eg.* neural networks) is not optimal. The best strategy would be to choose two policies as different from one another as possible.

4 Experimental results: combining handcrafted functions and neural networks

To analyse our method, we designed many individual policies to later combine them. These include:

- A handcrafted function based on heuristics, designed by human experts.
- Several fuzzy control functions.
- Conformant planning: a sequence of actions, independent of state observations.
- A one layer feedforward neural network with sigmoid activation functions, such that for a state at time t s_t, the action a_t is:

$$a_t = W_0 + W_1 \times tanh(W_2 \times s_t + W_3)$$

  Where W_0 is a bias vector of size the number of actions at each time step, W_1 the activation weights of the neurons in the hidden layer of dimension $\|actions\| \times \|neurons\|$, W_2 is the weight matrix from the states to the neurons of dimension $\|neurons\| \times \|states\|$, and W_3 is a bias vector the size

of the number of neurons. The total number N of parameters to optimise is then

$$N = \|states\| \times \|neurons\| + \|neurons\| \\ + \|actions\| \times \|neurons\| + \|actions\|$$

Fuzzy systems and conformant planning are intermediates between expert handcrafted functions and neural networks:

- They are less specialized than the expert function, which has only 3 parameters and works quite well.
- They are less parameter-free learners than the neural network.

Typically in our experiments the expert function is the best one for small learning time, and the neural network is the best function asymptotically. Interestingly however, we will see that our combination not only selects the best among the neural network and the expert function - it outperforms both.

4.1 Test problems: two types of unit commitment

Our test case is the one provided freely at `https://www.lri.fr/~teytaud/uctest/uctest.html`. In the unit commitment problem, the goal is to use available means of storage and production to satisfy a demand in energy over a given time horizon. We consider the case where energy can be produced from hydroelectric plants for free and from thermal plants at a cost. Energy can be stored until a certain limit in hydroelectric plants. The goal being to minimise the costs, we want to use the thermal plants as little as possible, and to maximise the efficiency of the storage available, while still meeting the demand. Failures to produce the required demand are heavily penalised.

We study our method on two distinct versions of this problem: a hydroelectric valley (all dams are connected in series), and a random network of dams avoiding cycles. In both cases, there are five dams, i.e. the state space contains 5 continuous variables. There are 21 time steps. Thermal units complete the dispatch, *ie.* they produce the electricity needed to satisfy the demand. In short, the control problem has 5 input variables, 5 output variables, and 21 time steps.

The dams receive random inflows at each time step, simulating weather conditions. We study two cases: rainy seasons with large inflows, and dry seasons with small inflows. The noise-free setting corresponds to a case in which we assume that all random processes can be predicted with high accuracy.The noisy setting represents a more difficult scenario. We need a noisy optimisation algorithm instead of a classical optimisation algorithm. In the noisy case, each fitness evaluation at iteration i of the evolutionary algorithm is averaged over $\lceil 10\sqrt{i} \rceil$ runs in order to mitigate the level of noise [1].

4.2 Noise free setting

We first present experiments in a simplified noise-free case, *ie.* the objective function is deterministic. This means that all random processes are replaced by a deterministic simplified counterpart. Results are presented in Fig. 1 (hydroelectric valley in the noise-free case; top: large inflows; bottom: small inflows) and Fig. 2 (hydroelectric network in the noise-free case, same two settings). In each of these four noise-free cases, the combination is at least as efficient as each policy separately, and in two cases it outperforms them vastly. Each experiment is reproduced with various numbers of neurons; 2 or 4 neurons is usually optimal.

4.3 Noisy setting

We now perform experiments with random noise around the mean inflows and demands. Fig. 3 presents the results in the case of the hydroelectric valley and Fig. 4 presents the results in the case of the hydroelectric random network (in both cases, two settings, namely large inflows and small inflows). In each of these four noisy cases, the combination is at least as efficient as each policy separately, and in two cases it outperforms them vastly. Each experiment is reproduced with various numbers of neurons; 2 or 4 neurons is usually optimal.

4.4 Experimental results: others

We also tried to replace the neural network policy by some other parametric policies such as fuzzy controllers, conformant planning, linear or quadratic controllers. However, none of them could be combined with the expert policy as efficiently as the neural network could.

Even more interestingly, when we combined two parametric policies, we could at best approximately get the best of the two (or four in cases of recursive combinations) but we never outperformed it. Furthermore, there was a clear delay to reach this selection, which is a result comparable to [10].

5 Conclusions and further works

We proposed a simple tool for combining parametric DPS policies:

- Just one optimisation pass for both policies (though we might consider more than two parametric policies);
- Usually quickly as good as the best of the considered policies;
- Sometimes much better.

Compared to separate learning, this makes the tool simpler (just one run) and faster (no separate learning). Compared to algorithm selection methods as [10], we can outperform both approaches, whereas classical algorithm selection can only be equivalent to the best of the two methods.

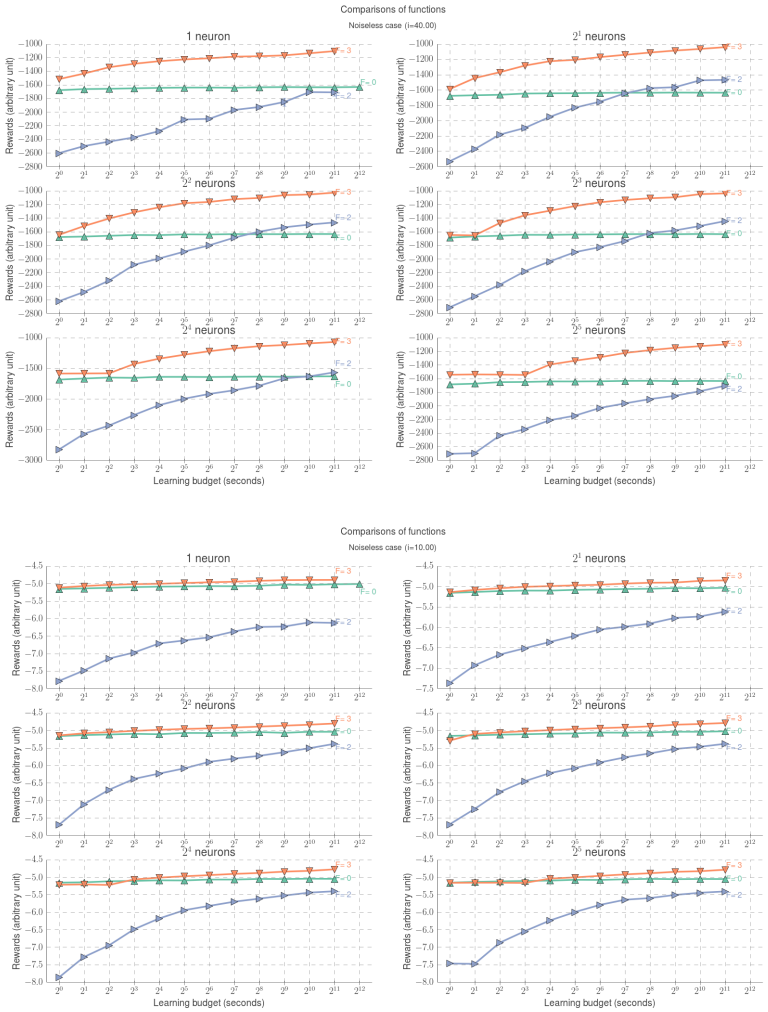

Fig. 1. Y-axis = reward. X-axis = learning budget. Hydroelectric valley. Noise-free setting (i.e. all random processes are simplified to their average values). Each subplot corresponds to a different number of neurons. △: parametric expert function. ▷: neural network. ▽: combination. The combination outperforms both separate functions. Top: large inflows. Bottom: small inflows.

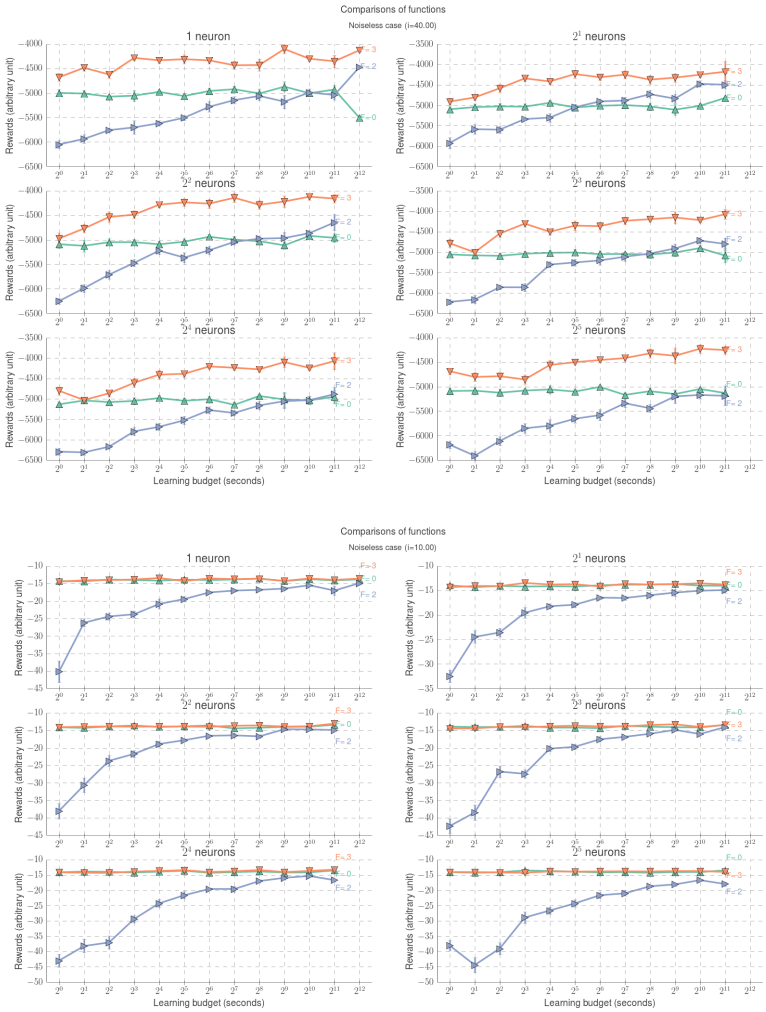

Fig. 2. Hydroelectric network, noise-free setting. Top: large inflows. Bottom: low inflows. The combination ($\triangleright$) is a clear success in this case as well, though in the latter case the expert function also performs very well.

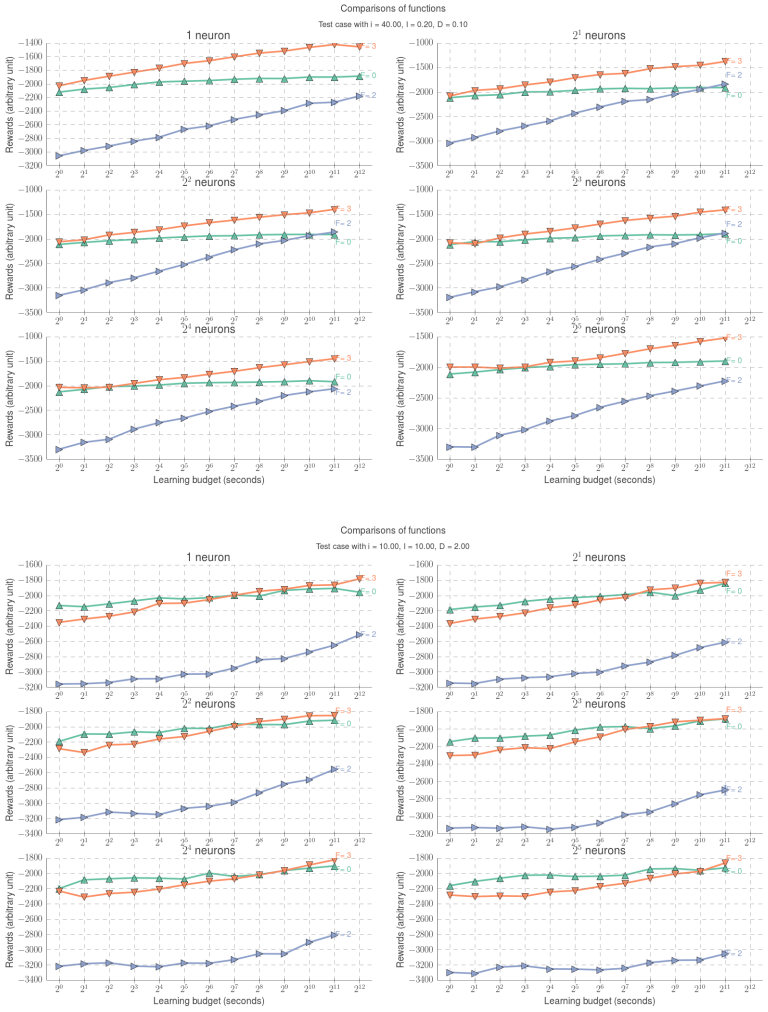

Fig. 3. Noisy setting, hydroelectric valley. Top: large inflow - the combination is excellent. Bottom: small inflows - the combination performs well; it does not always outperform the best of both solvers, but we point out that just selecting the best of two controllers takes more time than training them [10].

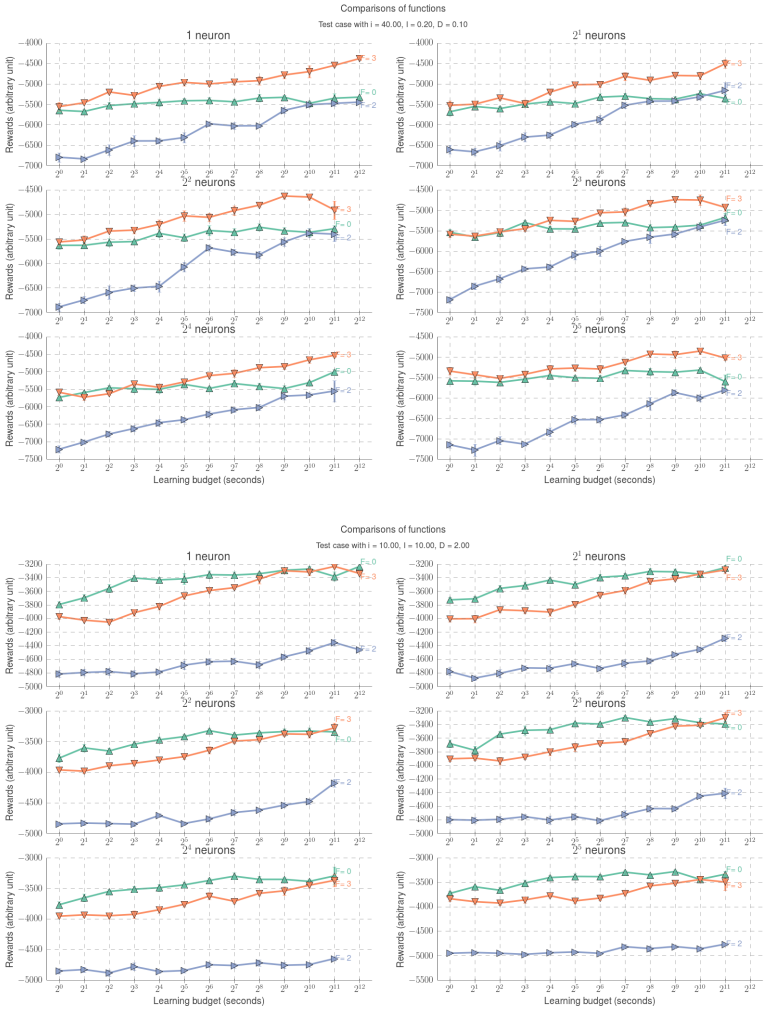

Fig. 4. Noisy setting, hydroelectric network. Top: large inflows. Bottom: small inflows. Results are qualitatively similar to Fig. 3.

We do not claim that we outperform portfolio methods, or at least not in all cases. Maybe for combining large numbers of policies our method would fail compared to portfolio methods. A limitation of our approach is that we can combine various parametric policies, but we can not combine DPS and completely different methods such as stochastic dual dynamic programming [18] or Monte Carlo Tree Search [8, 15]. Also, the success of our method was not reproduced with something else than the combination "expertise + neural networks"; we assume that this is related to the orthogonality (the expert policy is very different from the generic neural network). Still, the combination was very efficient in a stable manner, outperforming both methods without additional cost and without sophisticated developments. This was the case for $1, 2, 4, 8, 16, 32$ neurons, in all 8 sets of experiments (a deterministic and a stochastic case; a hydroelectric valley and a hydroelectric random network; and two levels of inflows). Therefore we consider that our simple combination (Eq. 2) should at least be considered when combining policies.

Last, we point out a specific property of evolution strategies. In the case where only one of the policies is relevant, then an optimization algorithm (evolutionary or not) might quickly find the optimal extreme value for α in Eq. 2. Then, the variables from the other policy have no impact on the objective function anymore, due to the weight zero of the corresponding policy. As a consequence, many variables become pointless, with no impact on the objective function. In contrast to many optimization algorithms, many evolutionary algorithms are not impacted by the presence of these pointless variables. Therefore, once α has been tuned, the evolutionary algorithm might just optimize the parameters of the relevant policy.

Combining four controllers was briefly considered in this work, without clear results. We considered combinations of controllers with less orthogonality (fuzzy systems, conformant planning, linear controllers) and results were far less convincing; whereas for neural networks and handcrafted policies the combination was already efficient. Extending the method in cases with less orthogonality might be interesting, as well as validating the fact that orthogonality is crucial.

References

1. Astete-Morales, S., Liu, J., Teytaud, O.: log-log convergence for noisy optimization. In: Proceedings of EA 2013. p. accepted. LLNCS, Springer (2013)
2. Astrom, K.: Optimal control of Markov decision processes with incomplete state estimation. Journal of Mathematical Analysis and Applications 10, 174–205 (1965)
3. Baudis, P., Posik, P.: Online black-box algorithm portfolios for continuous optimization. In: PPSN. pp. 40–49 (2014)
4. Bellman, R.: Dynamic Programming. Princeton Univ. Press (1957)
5. Bengio, Y.: Using a financial training criterion rather than a prediction criterion. CIRANO Working Papers 98s-21, CIRANO (1998), `http://ideas.repec.org/p/cir/cirwor/98s-21.html`
6. Beyer, H.G.: The Theory of Evolution Strategies. Natural Computing Series, Springer, Heideberg (2001)

7. Christophe, J.J., Decock, J., Teytaud, O.: Direct model predictive control. In: European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN). Bruges, Belgique (Apr 2014), http://hal.inria.fr/hal-00958192
8. Coulom, R.: Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In P. Ciancarini and H. J. van den Herik, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy pp. 72–83 (2006)
9. Doya, K., Samejima, K.: Multiple model-based reinforcement learning. Neural Computation 14, 1347–1369 (2002)
10. Gagliolo, M.: Online Dynamic Algorithm Portfolios. Ph.D. thesis, IDSIA/University of Lugano, Lugano, Switzerland (March 2010), http://como.vub.ac.be/~mgagliol/Gagliolo10PhD.pdf
11. Hamadi, Y.: Search: from Algorithms to Systems (HDR). Habilitation à diriger des recherches, Université Paris-Sud (2013)
12. van Hasselt, H.P.: Insights in Reinforcement Learning: formal analysis and empirical evaluation of temporal-difference learning algorithms. Ph.D. thesis, Universiteit Utrecht (January 2011), http://homepages.cwi.nl/~hasselt/papers/Insights_in_Reinforcement_Learning_Hado_van_Hasselt.pdf
13. Heidrich-Meisner, V., Igel, C.: Hoeffding and bernstein races for selecting policies in evolutionary direct policy search. In: ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning. pp. 401–408. ACM, New York, NY, USA (2009)
14. Kleinman, N.L., Spall, J.C., Naiman, D.Q.: Simulation-based optimization with stochastic approximation using common random numbers. Management Science 45(11), 1570–1578 (1999), http://pubsonline.informs.org/doi/abs/10.1287/mnsc.45.11.1570
15. Kocsis, L., Szepesvari, C.: Bandit based Monte-Carlo planning. In: 15th European Conference on Machine Learning (ECML). pp. 282–293 (2006)
16. Marivate, V., Littman, M.: An ensemble of linearly combined reinforcement-learning agents (2013), https://www.aaai.org/ocs/index.php/WS/AAAIW13/paper/view/7025/6704
17. Nudelman, E., Leyton-Brown, K., Hoos, H.H., Devkar, A., Shoham, Y.: Understanding random sat: beyond the clauses-to-variables ratio. In: Wallace, M. (ed.) Principles and Practice of Constraint Programming CP 2004, LLNCS 3258. vol. 3258 of Lecture Notes in Computer Science, pp. 438–452. Springer Berlin / Heidelberg (2004)
18. Pereira, M.V.F., Pinto, L.M.V.G.: Multi-stage stochastic optimization applied to energy planning. Math. Program. 52(2), 359–375 (Oct 1991), http://dx.doi.org/10.1007/BF01582895
19. Powell, W.B.: Approximate Dynamic Programming. Wiley (2007)
20. Samulowitz, H., Memisevic, R.: Learning to solve qbf. In: Proceedings of the 22nd National Conference on Artificial Intelligence. pp. 255–260. AAAI (2007)
21. Stalph, P.O., Ebner, M., Michel, M., Pfaff, B., Benz, R.: Genetic and evolutionary computation conference, gecco 2008, proceedings, atlanta, ga, usa, july 12-16, 2008. In: Ryan, C., Keijzer, M. (eds.) GECCO. pp. 535–536. ACM (2008)
22. Strens, M., Moore, A.: Direct policy search using paired statistical tests. In: Proceedings of the 18th International Conference on Machine Learning. pp. 545–552. Morgan Kaufmann, San Francisco, CA (2001)
23. Strens, M., Moore, A., Brodley, C., Danyluk, A.: Policy search using paired comparisons. In: Journal of Machine Learning Research. pp. 921–950 (2002)

24. Zadeh, L.A.: The birth and evolution of fuzzy logic. Int. J. of General Systems pp. 95–105 (1990)
25. Zambelli, M., Soares Filho, S., Toscano, A.E., Santos, E.d., Silva Filho, D.d.: NEWAVE versus ODIN: comparison of stochastic and deterministic models for the long term hydropower scheduling of the interconnected brazilian system. Sba: Controle & Automacao Sociedade Brasileira de Automatica 22, 598 – 609 (12 2011), http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-17592011000600005&nrm=iso